28 FEVRIER 2019

PFE : THE INDEPENDENT SUPERVISOR

MANUEL DE DEVELOPPEUR | DII -2018/2019

BRANDON SIMON-VERMOT
ENCADRANT : M. AMEUR SOUKHAL

Commentaire	Date de modification
Création du document	27/02/2019
Ajout de l'introduction	27/02/2019
Ajout du contenu	28/02/2019

Table des matières

Introduction

Dans le cadre de notre formation en école d'ingénieurs, à Polytech'Tours, nous devons réaliser un projet de fin d'études. Pour cela plusieurs sujets nous sont proposés, projets proposés par les différents professeurs de l'école ou proposés par des entreprises. Cependant il est aussi possible de proposer un projet personnel, ce qui est mon cas.

Pour proposer un projet, il faut au préalable fournir un cahier des charges et que le sujet soit étudié puis validé par un jury, tout cela se passe avant le début de l'année scolaire. Ainsi le sujet est évalué afin de déterminer s'il répond aux critères attendus.

Le contenu de ce manuel concerne les développeurs de l'application Web, les personnes qui reprendront le projet. Ce qui leur permettra d'obtenir le plus d'informations sur le fonctionnement de l'application, de ce qui est existant.

Prérequis

NodeJS

Au cours de ce projet, la partie Back-End est représentée par l'utilisation de NodeJS. Pour télécharger la dernière version, sachant qu'on utilise un système d'exploitation Linux, dirigez-vous directement sur le site officiel de NodeJS.

Démarrer le serveur NodeJS peut correspondre seulement à la partie Back-End mais aussi il est possible de regrouper la partie Front-End et la partie Back-End en une application, ce qui a été utilisé au cours de ce projet, de cette manière cela nécessite qu'un seul port. On utilise la commande suivante pour démarrer le serveur :

```
> npm run start:server
```

On peut retrouver cette commande dans le fichier de configuration « package.json ».

Angular CLI

La partie Front-End sera couverte par l'utilisation d'Angular, ne pas confondre avec AngularJS, ici nous utilisons la version utilisant du TypeScript. Pour télécharger Angular CLI, il est possible de passer par le gestionnaire de paquets de NodeJS, npm.

```
>  npm install -g @angular/cli
```

Il est possible de démarrer indépendamment la partie Front-End via la commande :

```
> ng serve
```

Visual Studio Code

L'utilisation de l'IDE Visual Studio Code est un choix parmi tant d'autres, cela ne provoque en aucun cas un problème d'utiliser un autre IDE. Pour installer la dernière version de Visual Studio Code, elle est disponible sur le site officiel.

Grâce à cet IDE, il est possible de télécharger facilement des extensions, qui permettront de rendre l'interface plus esthétique mais aussi cela permettra de reconnaitre les éventuels problèmes de syntaxe.

L'installation de l'extension « Angular Essentials » est recommandée pour améliorer l'expérience d'utilisation de Visual Studio Code avec Angular, elle contient plusieurs extensions en un seul paquet.

Optionnellement, installer l'extension « Material Icon Theme » permet de rendre les différentes icônes de dossiers et fichiers d'un projet dans Visual Studio Code, plus esthétique.

Explications sur l'architecture du projet

Plusieurs fichiers de configuration sont disponibles à la racine du projet. Le fichier « package.json » contient toutes les dépendances mais aussi toutes les dépendances disponibles seulement pour le développement de l'application. Toutes ces dépendances sont installées dans le dossier « node_modules ».

Les fichiers angular.json et tsconfig.json correspondent respectivement à la configuration d'Angular CLI et à la configuration du TypeScript, les valeurs par défaut n'ont pas été modifiées.

Le dossier « e2e » correspond aux tests End-to-End mais aussi le dossier « backend » qui contient les fichiers sources de la partie Back-End et le « src » contient les fichiers sources de la partie Front-End du projet.

Dans le dossier « dist » il est possible de retrouver la partie Front-End compilé, dans le but de déployer deux applications indépendantes, cependant dans le contexte du projet, la partie Angular se retrouve obsolète sans la partie Back-End, ce qui rend inutile de déployer deux applications, puisqu'il n'y a qu'un seul de serveur dédié disponible actuellement.

Back-End

Le dossier de la partie lié à NodeJS, il contient aussi la partie Front-End compilé dans le dossier « angular », afin de permettre de déployer une seule et même application.

Un fichier « package.json » est aussi présent afin de séparer correctement les dépendances d'Angular et celle de NodeJS. Un autre fichier lié à la configuration est présent à la racine de ce dossier, celui-ci nommé « server.js » correspond à la configuration du serveur NodeJS, cela comprend donc la création du serveur mais aussi l'attribution du port d'écoute.

Le dernier fichier présent à la racine, « app.js » correspond au « main » de notre application Back-End, celui-ci regroupe les différents imports qui seront nécessaires à notre application, la connexion à la base de données mais aussi la définition générique des différentes routes.

Cependant, elles seront détaillées ainsi que les méthodes http qui sont disponibles dans un fichier propre qui sera présent dans le dossier « routes », ainsi il est possible d'associer une route et une méthode http avec une fonction qui sera créée dans un fichier regroupant toutes les fonctions d'un contrôleur, dans le dossier « controller ».

Dans ce dernier, nous allons pouvoir avoir toute la partie logique de l'application côté Back-End. Pour valider la conformité des données qu'on souhaite ajouter ou modifier dans notre base de données, malgré l'utilisation d'une base NoSQL, il y a un modèle à respecter afin d'assurer que les champs requis sont bien présents. Ces modèles sont enregistrés dans un fichier propre à chaque contrôleur, présent dans le dossier « models ».

Le dossier « middleware » contient des fonctions de vérification, dans notre cas nous avons pour le moment deux fichiers permettant de vérifier si un utilisateur est connecté avant de réaliser quelconque action nécessitant une connexion mais aussi un second fichier qui lui va vérifier si le fichier qu'on souhaite enregistrer sur le serveur correspond bien à des règles (type, taille, etc.).

Le dossier « scripts » contient les scripts permettant de lancer, d'arrêter ou de mettre à jour une application de notre parc d'application. Quant au dossier « images » celui-ci contient les images qui sont liées à une suggestion ou une application.

Front-End

Pour éditer le contenu de l'application visible par les utilisateurs, tout se trouve dans le dossier « src » à la racine du projet. Celui-ci contient plusieurs fichiers de configuration qui n'ont pas été modifiés mais aussi l'icône de l'application Web « favicon.ico » qui sera visible dans l'onglet du navigateur Web de votre utilisateur.

Le fichier « index.html » contient le titre de l'application Web visible comme l'icône dans l'onglet du navigateur Web de votre utilisateur. On définit ainsi que le corps de notre application contiendra un composant « app-root » qui correspond à notre composant principal créé grâce à Angular. On l'appelle comme si c'était une balise HTML classique et ça sera le cas pour tous nos sous-composants qui constitueront notre principal.

Il est possible de configurer un thème sombre et/ou clair pour notre application via le fichier « tib-theme.scss ».

Au cours du développement, il peut s'avérer utile d'utiliser des variables d'environnements, pour l'environnement de production mais aussi disponible seulement en environnement de développement. Ces variables sont écrites dans deux différents fichiers disponibles dans le dossier « environments ». L'environnement actuel est détecté par le fichier « main.ts » disponible à la racine du dossier « src ».

Le dossier le plus important est le dossier « app » contient tous les imports de librairie utile à l'application via le fichier principal « app.module.ts » mais il possible de diviser le fichier et appeler les sous-fichiers depuis le fichier principal afin de le rendre plus lisible et ainsi regrouper les imports nécessaires pour utiliser par exemple la librairie « Angular Material », via le fichier « angular-material.module.ts » .

Le fichier « app-routing.module.ts » regroupe toutes les routes disponibles pour notre application Web, liant ainsi une ou plusieurs routes à un composant, avec la possibilité d'ajouter une vérification d'authentification, limitant ainsi certaines actions aux utilisateurs authentifiés ou encore aux administrateurs. Dans le cas d'une route non définie, il est possible de rediriger nos utilisateurs vers une page spécifique.

Pour chaque composant un dossier propre est présent à la racine du dossier « app », qui dans le format le plus simple contient un fichier html pour le contenu de notre composant, un fichier css pour le style, un fichier « xxx.component.ts » pour la partie « logique » de notre composant, permettant ainsi de rendre dynamique notre contenu HTML. Il est possible de retrouver un fichier « xxx.component.spec.ts » qui lui contient les différents tests qui sont appliqués à notre composant afin de le valider.

De la même manière que pour notre partie Back-End, nous pouvons avoir un fichier « xxx.component.model.ts » qui va permettre d'exporter un type de variable afin de définir un modèle de données pour éviter qu'un souci soit présent dans les données attendues par le Back-End.

Pour communiquer avec la partie Back-End, un fichier « xxx.service.ts » doit être présent permettant de définir toutes les fonctions qui vont faire appel à une route et une méthode http disponible côté Back-End et ainsi ajouter, modifier, supprimer ou encore obtenir des informations.

Ceci est pour un composant simple, un seul dossier contenant tous ces fichiers. Pour un composant plus complexe, un dossier peut ranger plusieurs dossiers qui correspondent à des composants qui sont liés, comme le composant pour l'ajout ou la modification d'une suggestion et le composant pour lister les suggestions, tous deux sont liés aux données autour des suggestions, ainsi en rangeant ces deux composants dans un seul et même dossier, nous pouvons plus facilement nous y retrouver.

Cependant le fonctionnement est le même qu'un composant simple, il contient simplement une couche supplémentaire, par exemple dans le cas des composants liés aux suggestions, nous avons un fichier « posts.module.ts » qui va lier les composants « post-create » et « post-list » qui sont tous deux des composants « simple ». Les deux dossiers de ces deux composants sont donc présents dans le dossier « posts », ainsi que le fichier « posts.module.ts ». D'autres fichiers tels que le fichier « xxx.service.ts » et « xxx.module.ts » sont présents dans la couche supérieure puisque ce sont des fonctions et exports partagées.

Explications de l'API

Pour chaque route spécifié dans cette catégorie, une fonction côté Angular, présente dans un fichier « xxx.service.ts » contenu dans le dossier du composant en question. Cette fonction fait appel à une route définie dans le contrôleur associé présent dans le dossier « routes » côté NodeJS. Chaque route est liée à une fonction définie dans le fichier du contrôleur associé, présent dans le dossier « controllers » toujours côté Back-End.

Pour chaque route de l'API, l'utilisateur doit être authentifié. Cependant certaines fonctions sont limitées et disponibles seulement pour les administrateurs.

Utilisateurs

Utilisateurs	
GET	/api/users
GET	/api/users/:id
POST	/api/users/signup
POST	/api/users/login

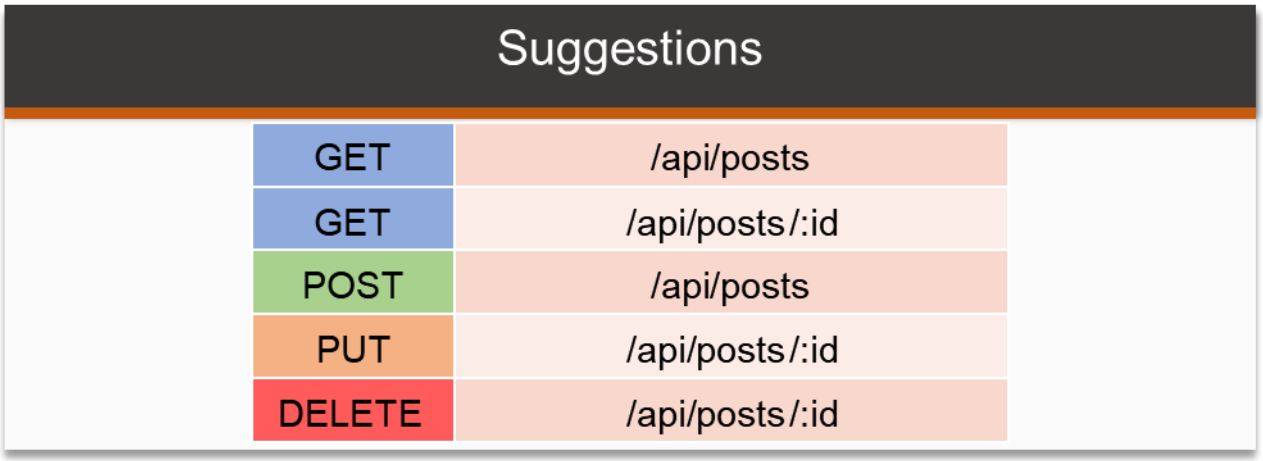
1. La première méthode GET, permet d'obtenir une liste de tous les utilisateurs qui sont actuellement inscrits sur le site Web, ainsi que des informations associées. Cette route n'est disponible que pour les administrateurs.

2. Pour obtenir plus d'informations sur un utilisateur, l'utilisation de la même route que pour lister les utilisateurs en associant un id à la fin de l'URL est nécessaire. Cette route n'est disponible que pour les administrateurs.

3. Pour qu'un visiteur puisse s'inscrire, la méthode POST avec la route « /api/users/signup » est appelée, par la suite, cette route sera supprimée, limitant ainsi l'accès au site Web et à ces fonctionnalités.

4. Une fois inscrit, le visiteur pourra s'authentifier ainsi la route « login » sera appelée, permettant de vérifier si les informations de l'utilisateur sont correctes. Un token JWT sera généré puis transmis en retour pour le stocker dans l'espace de stockage local du navigateur.

Suggestions



1. & 2. Le principe de fonctionnement est exactement le même que pour les utilisateurs, le seul changement présent est la cible, ici nous voulons obtenir les informations des suggestions ou d'une suggestion.

3. L'ajout d'une suggestion fait appel à la route « /api/posts » avec la méthode POST, cela permet de sauvegarder en base de données les informations liées à une suggestion.

4. Pour sauvegarder les modifications d'une suggestion, la route avec la méthode PUT est appelée. Cette route est disponible seulement pour le créateur de la suggestion et les administrateurs.

5. Il est possible de supprimer une suggestion via la méthode DELETE. Cette route n'est disponible seulement pour le créateur de la suggestion et les administrateurs.

Applications

Applications	
GET	/api/apps
GET	/api/apps/:id
POST	/api/apps
PUT	/api/apps/:id
PUT	/api/apps/:id/run
PUT	/api/apps/:id/stop
PUT	/api/apps/:id/update
DELETE	/api/apps/:id

1. , 2., 3., 4., & 8. Ces routes reviennent au même principe que pour les suggestions, cependant ces routes correspondent aux applications du parc. Seules les routes associées à une méthode GET seront disponibles pour les utilisateurs, les autres routes sont réservées aux administrateurs.

5. Cette route permet de lancer une application en mode simple, ainsi si l'application s'arrête, elle ne sera pas redémarrée. Le script « run.sh » associé à l'application sera exécuté. Cette route n'est disponible que pour les administrateurs.

6. Si une application est en cours d'exécution quel que soit son mode de lancement, il sera possible de l'arrêter via cette route. Pour le moment, l'application est tuée, par la suite le script « stop.sh » associé à l'application sera exécuté, dans le but d'éteindre proprement l'application, évitant par la même occasion de brusquer les utilisateurs. Cette route est disponible seulement pour les administrateurs.

7. Si une application est arrêtée, il est possible d'exécuter le script « update.sh » via cette route, permettant de mettre à jour l'application. Cette route n'est disponible que pour les administrateurs.

Statistiques

1. Seuls les administrateurs peuvent enclencher l'appel de cette route, permettant ainsi d'obtenir les informations de performances du serveur dédié. Par défaut, toutes les 10 minutes, des nouvelles informations sont récupérées et stockées en base de données.

Comment modifier l'intervalle de temps pour l'acquisition de données ?

Le changement de données lié à la sauvegarde d'informations sur les performances du serveur dédié, se passe côté Back-End, autrement dit côté NodeJS. Dans le dossier « controllers » se trouve un fichier « dataServer.js » qui contient toutes les fonctions associées aux données du serveur.

Dans ce fichier après les différents imports se trouve la déclaration d'une constante « interval », sa valeur est en millisecondes et indique le temps entre chaque extraction de données et enregistrement en base de données.

```
const interval = 60000 ;
```

Modifier cette valeur, permet donc de réduire ou d'augmenter l'intervalle de temps, par défaut la valeur correspond à 10 minutes d'intervalle entre chaque extraction de données.

Comment modifier le seuil de surcharge ?

Le changement de données lié à la définition des règles d'arrêt pour les applications en cours d'exécution, se passe côté Back-End, autrement dit côté NodeJS. Dans le dossier « controllers » se trouve un fichier « dataServer.js » qui contient toutes les fonctions associées aux données du serveur.

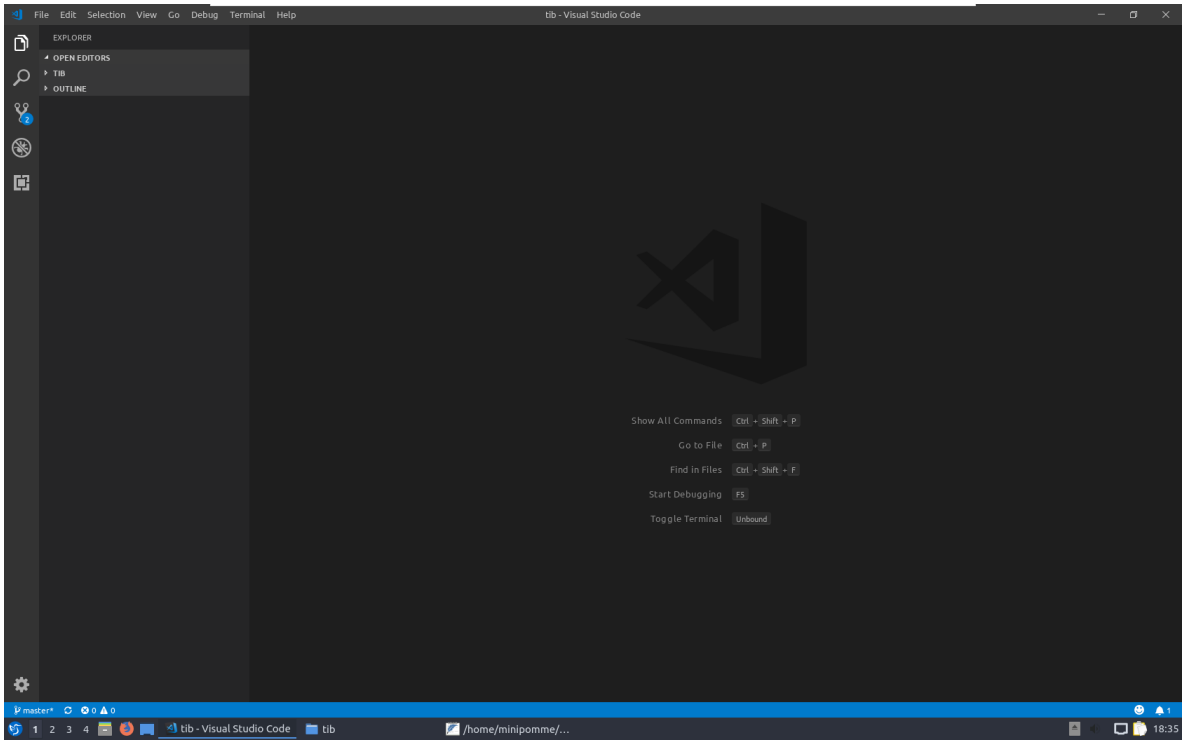
Dans ce fichier après les différents imports se trouve la déclaration d'une constante « limit », sa valeur est en pourcentage et indique le seuil qu'une donnée (taux d'utilisation du CPU, taux d'utilisation de RAM, taux d'utilisation du disque) ne dépasse pas ce seuil.

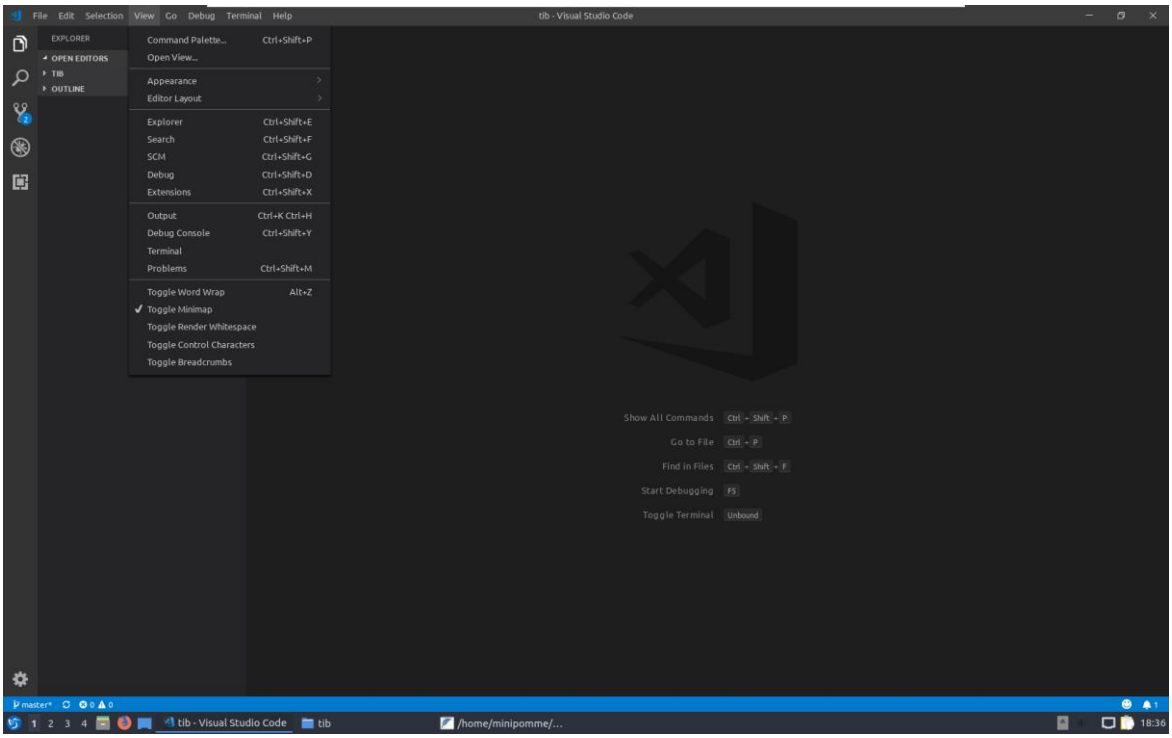
```
const limit = 80.0 ;
```

Modifier cette valeur, permet donc de réduire ou d'augmenter le seuil qui ne doit pas être dépassé pour l'une des valeurs du serveur dédié, par défaut la valeur correspond à 80%. Dans le cas échéant, le serveur dédié est considéré en surcharge, le fonctionnement actuel de cette règle, arrête la dernière application lancée. Par la suite, d'autres règles s'ajouteront pour déterminer l'application qui demande le plus de performances mais qui possède le moins d'utilisateurs actifs afin d'arrêter l'application ayant le moins d'impact sur le système. Limitant ainsi le maximum de perturbation pour l'expérience de nos utilisateurs.

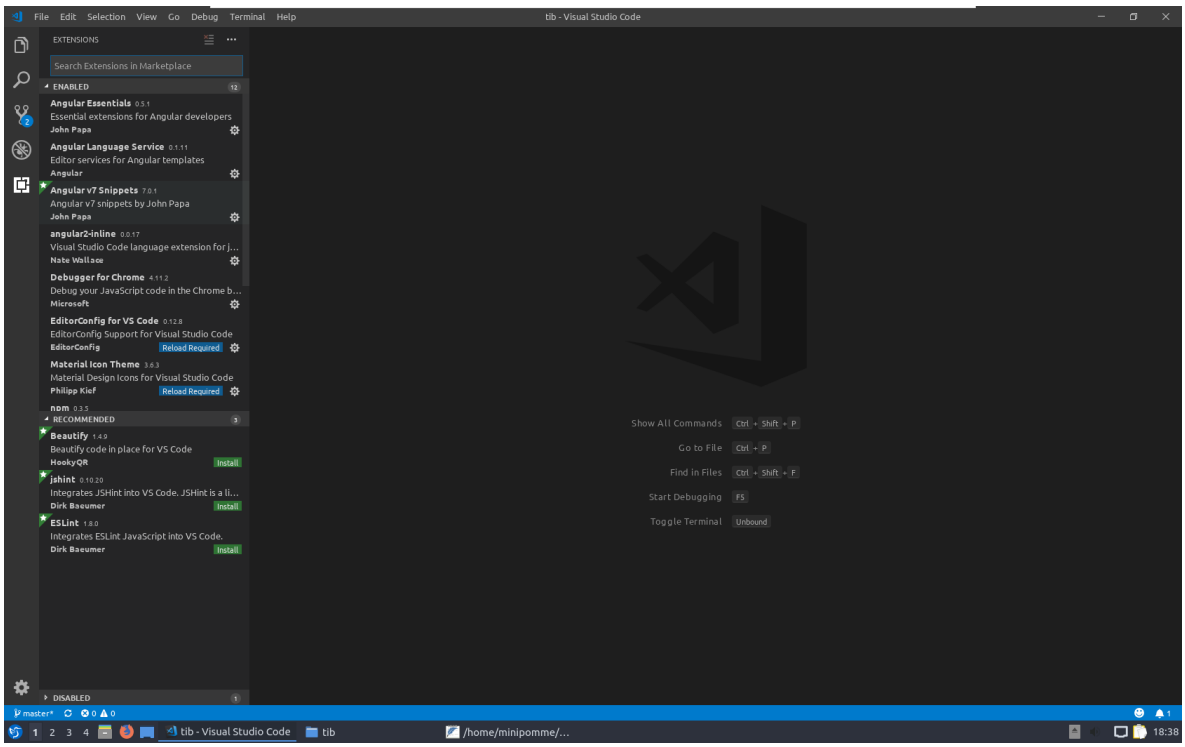
Comment installer une extension pour Visual Studio Code ?

Ci-dessous on peut retrouver l'interface de Visual Studio Code, une interface sobre qui permet d'éviter de trouver rapidement et facilement ce que l'on souhaite. Pour télécharger des extensions afin de les ajouter à notre IDE, il faut cliquer sur l'onglet « View », puis sur « Extensions ».





Un onglet s'ouvre indiquant toutes les extensions installées et activées, il est aussi possible de retrouver en bas de l'onglet toutes les extensions que l'IDE nous recommande (par exemple suite à l'installation d'Angular CLI, il est possible d'avoir des extensions qui sont en lien avec Angular), elles sont visibles grâce à une petite étoile qui se trouve au-dessus des noms des extensions. Une barre de recherche est cependant disponible afin de trouver exactement l'extension que l'on recherche.



Après l'installation d'une extension, il est important de redémarrer l'IDE afin qu'il termine l'installation et active votre extension.

Comment installer une dépendance ?

Pour l'installation d'une dépendance liée à Angular ou NodeJS, l'utilisation du gestionnaire de paquets de NodeJS, npm est recommandée. L'utilisation de cette commande se fait de deux façons :

```
npm install [-options] <name>
```

Ou par son alias:

```
npm i [-options] <name>
```

Par défaut, l'installation ne se fait que sur votre machine, les dépendances ne seront pas enregistrées dans le projet. Il est possible cependant de sauvegarder ces dépendances pour que le mainteneur puisse déployer correctement le système.

Pour sauvegarder cette dépendance pour la version de production :

```
npm install --save-prod <name>
```

Ou par son alias :

```
npm install -P <name>
```

Cependant pour des dépendances liées au développement, aux tests, il est important de ne pas les sauvegarder en production, mais les garder seulement pour la version de développement :

```
npm install --save-dev <name>
```

Ou par son alias :

```
npm install -D <name>
```

Comment compiler la partie Front-End ?

Dans le but de déployer son application, il est possible de lancer deux applications qui vont gérer indépendamment la partie Front-End et la partie Back-End ou dans notre cas, déployer une seule et même application qui gère les deux parties.

Par défaut, la compilation du TypeScript se fait dans le dossier « dist » à la racine du projet, mais il est possible de modifier cela en réglant le chemin cible dans le fichier de configuration « angular.json » à la racine du projet.

À l'intérieur de ce fichier, nous pouvons voir l'objet « projects » qui contient l'objet « tis » qui correspond au nom du projet, défini lors de la création de l'application via Angular CLI. Cet objet lui aussi contient d'autres objets, ici nous nous intéressons à l'objet « build » et notamment à l'objet « options » qui contient l'option « outputPath ».

La valeur associée à cette option, détermine le chemin dans lequel l'application sera compilée, prête à être déployée. Pour compiler l'application cela se fait par la commande suivante :

```
ng build --prod
```